Ground Rules for Sustainable Code

Refactoring

Amr Noaman
Co-founder and Principal, Agile Academy
Amr Noaman Abdel-Hamid

Co-initiator of Egypt’s GoAgile program at 2011, to boost lean & agile software development in Egypt

Agile practitioner, coach, trainer, consultant, writer and lecturer

Co-founder of Agile Academy
Egypt Lean and Agile Network
Agile Egypt

ICAigle Authorized Instructor
SCEA, CSP, 6σ black-belt for IT

Author at international magazines (IEEE, Infoq), and speaker at global conferences (Agile Conference, PMI Global Congress)

Member of the Conference Program Team – Development Practices and Craftsmanship – Agile 2017

ICAigle Authorized Instructor
SCEA, CSP, 6σ black-belt for IT

Agile Alliance
ScrumAlliance

Email: amr@agileacademy.co
Blog: amr.agileegypt.org
Twitter: @AmrNoaman
A Story of a Great Team
What's in common?

What's different?

What's in common?
Patterns for Legacy Code Refactoring

• Whole-or-nothing
• As-per-the-book
• Technical hero
• Try-retry
It doesn’t Work!
Why refactoring fails?

- Vague & hazy objectives
- Failed to automate tests
- Technical Glut Trap
- Unsustainable Development Pace
- It’s none of the managers’ business!
• What do we want to enhance in the code?
• What are the priorities? Which enhancement is more important than which?
• What are the short term objectives?
• What indicates progress?
• When are we going to finish?
Refactoring or Automated Tests?
We are busy refactoring the product code for the benefit of all of us.

It is a highly technical stuff which you will not grasp, even if you tried hard.

We need to concentrate. Please do not keep nagging for status and end dates.

When we finish, we'll let you know.

Thank you for your understanding. 😊😊
Vague & hazy objectives

Failed to automate tests

Technical Glut Trap

It’s none of the managers’ business!

Unsustainable Development Pace
Technical Glut Trap

no control over technical changes in the code

Code changed

Changed applied

+ Ideas to change the code
Why not fix this issue as well.

Hmmm, what about applying this pattern. It may affect only several other modules.

We’d better refactor this module, it’s just the right time.
How development becomes unsustainably with refactoring?
Mainline (Trunk)

RC1

Project A

Refactoring Branch

Project B

Sub-task of project B

R1
Vague & hazy objectives

Failed to automate tests

Technical Glut Trap

Unsustainable Development Pace

It’s none of the managers’ business!
Ground Rules for Sustainable Code Refactoring

GROUND RULES BEFORE YOU START
Development Habits

1. Re-writing the whole from scratch is an easy and *incorrect* solution!
2. Automating pin-down tests are always a cheap and effective starting point
3. Remove dead and unused code first. There is no point of refactoring code of zero value
If this is your garage, what’s the first step in preparing it?
Yes! Removing clutter
Think ...

A man who was stressed out, recently divorced, depressed, drinking way too much alcohol and using drugs,

felt dramatically better after he got rid of 80% of his belongings.

Similarly, if this is your code; first, remove the clutter.
Development Habits

4. Regardless of the amount of duplication you already have, stop duplicating any new code

5. The boy scout rule: "Always leave the campground cleaner than you found it"
#1 Principle of Pragmatic Programming:

*Don't Repeat Yourself!*

“Duplication may be the root of all evil in software”

“Duplicate code is the root of all evil in software design”

#1 deadly Sin by developers

- SonarQube
What’s Evil about Code Clones?

Out of the $100 Billion spent on software in the 90’s, $70 Billion are spent on Maintenance.

Cost of Maintenance: 70%
Cost of New Development: 30%

Cost of New Development
30%

Cost of Maintenance
70%

Drill down of the cost of maintenance

60% Cost of locating bugs

40% Everything else!

Fixing, testing, reviews, integration, system testing, deployment, user acceptance, ...
CM Environment

7. Enable work-item tracking for refactoring effort.
   – This invites geek developers to think a bit before they fall into the "technical glut trap"

8. Enable end-to-end traceability.
   – This will increase your visibility of which code written for which purpose
Traceability – Is It Really Worth it?

• Impact Analysis
Traceability – Is It Really Worth it?

• Root-cause Analysis
CM Environment

9. Avoid dedicating a branch for refactoring different from the mainline.
   – The longer the period of separation, the higher the probability being abandoned and never merge back

10. Enable and enforce the practice of deploying small, stepwise, and frequent improvements into production.
    – It feels scary only for the first couple of times
Mainline (Trunk)

- Project A
- Project B
  - Sub-task of project B
  - Refactoring Branch

RC1

R1
Planning & Tracking

11. Make refactoring results visible for everyone, including sponsors and senior managers

12. Dedicate an agreed upon percentage of team effort to refactoring; preferably not more than 20-30%

13. Never stop developing new feature or solving known bugs because you're busy refactoring. This will fire back very soon

14. Engage busy managers with frequent and up-to-date reports on code improvement metrics. Managers will not sponsor an activity which they cannot track or control
“Burn charts have become a favorite way to give visibility into a project’s progress. They are extremely simple and astonishingly powerful”

-- Alistair Cockburn
Test coverage (After 4 iterations) = 58% (103k / 176k)
Tracking Code Size Reduction

Code Size Reduction Target (CSR Target) =
  • 100% of Dead Code
  • 90% of Exact Clones
  • 60% of Similar Clones
Number of Violations burn-down

Number of methods with LOC > 10
Size Reduction Speed Run Chart

- Number of LOC removed per hour of work
- Calculated every iteration
Use them to engage busy managers and gain their support.
The First Rule of Managing Refactoring Effort:

Managers will not sponsor any activity unless they can track and control it
Ground Rules for Sustainable Code Refactoring

SUSTAINABLE REFACTURING ROADMAP
Quick Wins

- Remove dead code
- Remove code duplicates
- Reduce method size
- Enhance identifier naming

Divide & Conquer

- Split code into modules and reduce coupling
- Discover and grow physical and replaceable components
- Promote components to services or microservices

Inject Quality In

- Cover components with automated tests
- Enhance components internal design

Sustainable Refactoring

Roadmap v4.0

Continuous Inspection

©2014-2018 – Agile Academy – www.agileacademy.co
Quick Wins

- Remove dead code
- Remove code duplicates
- Reduce method size
- Enhance identifier naming

Divide & Conquer

- Discover & split code into components
- Optimize components’ public interfaces
- Reduce coupling

This Stuff is Risky Isn’t It?
No, It’s Not!

<table>
<thead>
<tr>
<th>Metric</th>
<th>Release 5.5 (Before refactoring)</th>
<th>Release 5.6 (During Refactoring)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total bugs detected</td>
<td>128</td>
<td>176</td>
</tr>
<tr>
<td>% of Regression bugs</td>
<td>29.7%</td>
<td>25.1%</td>
</tr>
<tr>
<td>Average bug fixing cost (hours)</td>
<td>1.97</td>
<td>1.8</td>
</tr>
</tbody>
</table>
A step by step guide to clean coding techniques and professional development habits

https://leanpub.com/RefactortingToCleanCode